**《计算机组成原理实验》指导手册**

实验二 寄存器堆与队列

1. 实验目标
   1. 掌握寄存器堆（Register File）和存储器（Memory）的功能、时序及其应用；
   2. 熟练掌握数据通路和控制器的设计和描述方法。
2. 实验内容

1. 寄存器堆

设计参数化的寄存器堆，其逻辑符号如图-1所示。该寄存器堆含有32 个寄存器（r0 ~ r31，其中r0的内容恒定为零），寄存器的位宽由参数WIDTH指定，具有2个异步读端口和1个同步写端口。

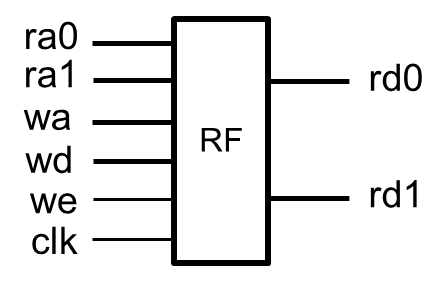


图-1 寄存器堆逻辑符号

参数化的寄存器堆端口声明如下：

module register\_file //32 x WIDTH寄存器堆

#(parameter WIDTH = 32) //数据宽度

(clk, //时钟（上升沿有效）

input [4:0] ra0, //读端口0地址

output [WIDTH-1:0] rd0, //读端口0数据

input [4:0] ra1, //读端口1地址

output [WIDTH-1:0] rd1, //读端口1数据

input [4:0] wa, //写端口地址

input we, //写使能，高电平有效

input [WIDTH-1:0] wd //写端口数据

);

……

endmodule

2. 存储器

存储器与寄存器堆的功能类似，都是用于存储信息，只是存储器的容量更大，配置方式更多，例如ROM/RAM、单端口/简单双端口/真正双端口、分布式/块式等方式。设计存储器可以通过行为方式描述，也可以通过IP例化方式实现。

例如，设计一容量为16 x 8位（即深度DEPTH：16，宽度WIDTH：8）的单端口RAM，其逻辑符号如图-2所示。用行为方式描述的Verilog代码如下：**（请补充代码中空缺的参数）**

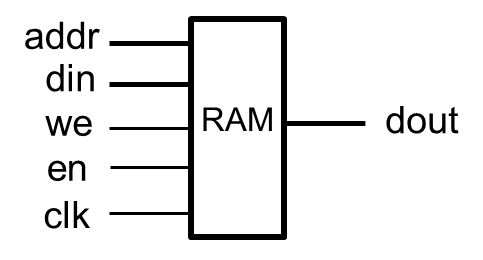


图-2 单端口RAM逻辑符号

module ram\_16x8 //16x8位单端口RAM

(input clk, //时钟（上升沿有效）

input en, we, //使能，写使能

input [ \_\_\_\_\_ ] addr, //地址

input [ \_\_\_\_\_ ] din, //输入数据

output [ \_\_\_\_\_ ] dout //输出数据

);

reg [ \_\_\_\_\_ ] addr\_reg;

reg [ \_\_\_\_\_ ] mem[ \_\_\_\_\_ ];

//初始化RAM的内容

initial

$readmemh(“初始化数据文件名”, mem);

assign dout = mem[addr\_reg];

always@(posedge clk) begin

if(en) begin

addr\_reg <= addr;

if(we)

mem[addr] <= din;

end

end

endmodule

Verilog HDL程序可以利用两个系统任务$readmemb和$readmemh，从文件中读取数据来初始化存储器。其语法格式（以$readmemb为例，$readmemh类似）如下：

$readmemb("<数据文件名>",<存储器名>,<起始地址>,<终止地址>);

数据文件是文本格式，只能包含空白（空格、换行、制表格tab）、注释和二进制（对于$ readmemb）或十六进制数据（$readmemh），数据中可以有不定值x或X、高阻值z或Z、或者下画线，但不能包含位宽书名和格式说明。起始地址和结束地址是可选的。当地址出现在数据文件中时，其格式是字符“@”后跟上十六进制数据，例如：@hhhh。当读取中遇到地址说明符，会将地址后的数据存放到相应的地址中。

例如，从文件“rx.vec”中读取的第一个数字被存储在地址15中，下一个存储在地址16，并以此类推直到地址30。

$readmemb("rx.vex", MemA, 15, 30);

例如，文件init.vec内容如下：

@002

11111111  01010101

00000000  10101010

@006

1111zzzz  00001111

Verilog程序如下：

reg [7:0] mem[0:7];

initial $readmemb("init.vec", mem);

则存储器中的内容如下：

mem[0]=xxxxxxxx;

mem[1]=xxxxxxxx;

mem[2]=11111111;

mem[3]=01010101;

mem[4]=00000000;

mem[5]=10101010;

mem[6]=1111zzzz;

mem[7]=00001111;

通过Vivado例化存储器IP的配置界面如图-3~8所示。配置时可以指定COE文件对存储器的内容初始化。COE文件包含两个分号结束的参数：memory\_initialization\_radix和memory\_initialization\_vector。前者说明数据进制，可以是2、10、16；后者列举出由空格或逗号分隔数据序列。

例如，初始化一个32x16 ROM的COE文件如下：

memory\_initialization\_radix = 16;

memory\_initialization\_vector =

23f4, 0721, 11ff, ABe1, 0001, 1, 0A, 0,

23f4, 0721, 11ff, ABe1, 0001, 1, 0A, 0,

23f4, 721, 11ff, ABe1, 0001, 1, A, 0,

23f4, 721, 11ff, ABe1, 0001, 1, A, 0;

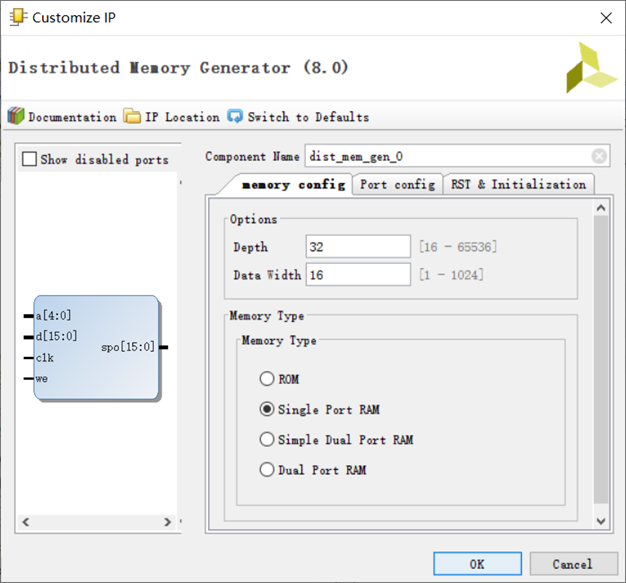


图-3 分布式存储器例化界面—存储器配置

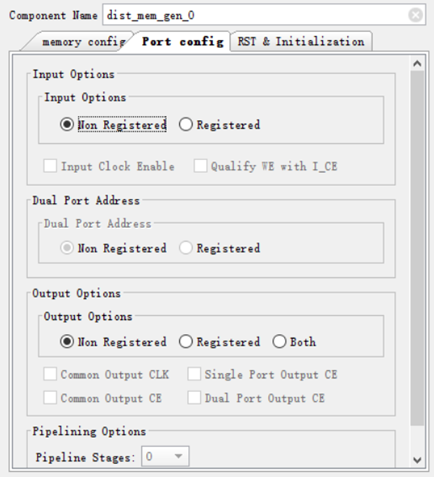


图-4 分布式存储器例化界面—端口配置

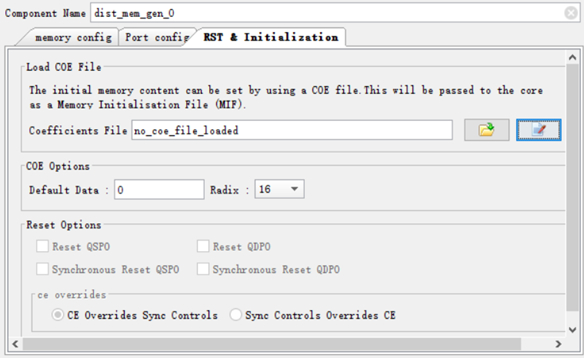


图-5 分布式存储器例化界面—复位和初始化

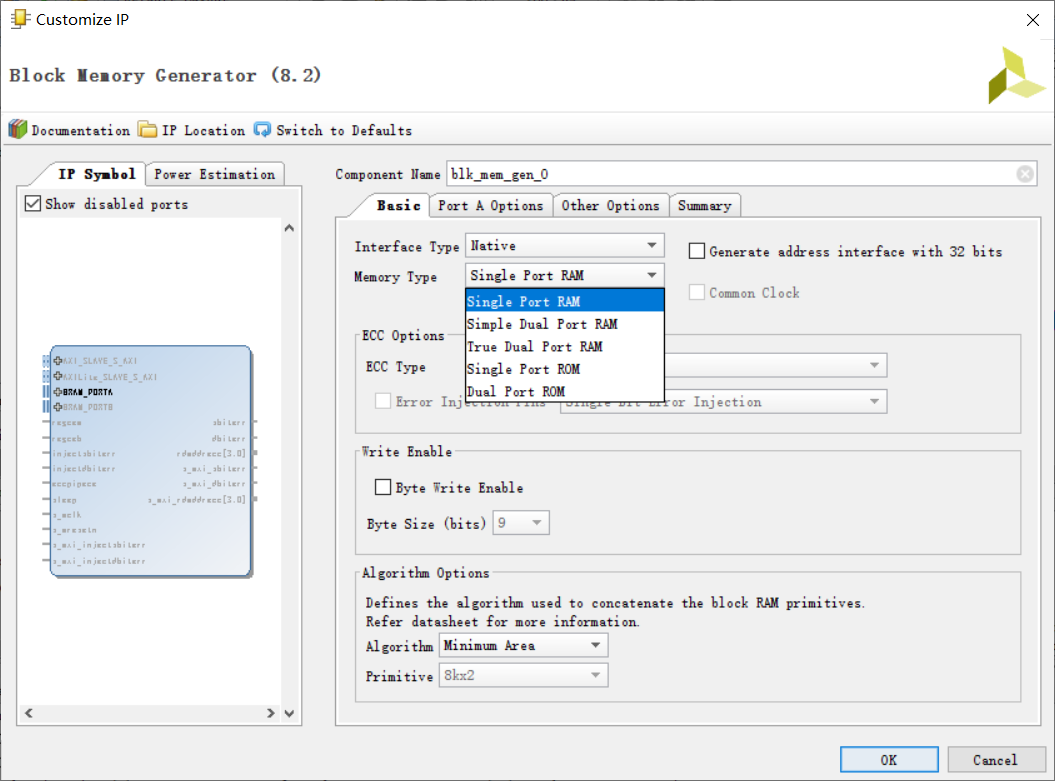


图-6 块式存储器例化界面—基本

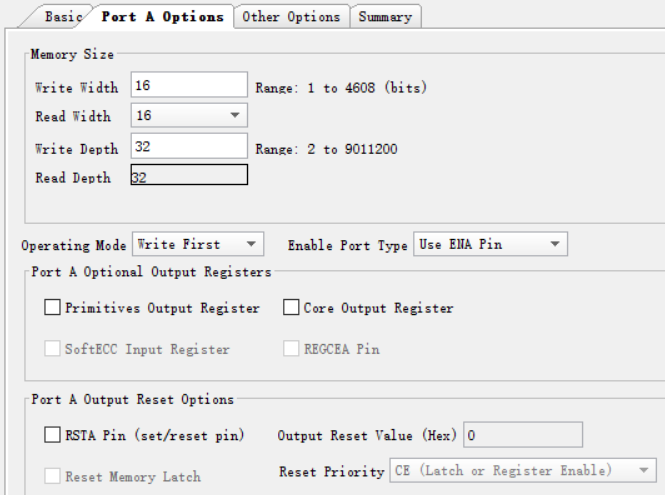


图-7 块式存储器例化界面—端口选项

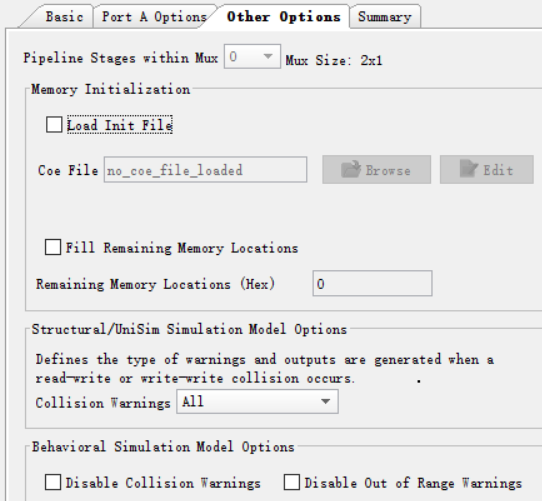


图-8 块式存储器例化界面—其他选项

3. 先进先出（FIFO）队列

利用例化的存储器IP（16 x 8位块式的单端口RAM）和适当的逻辑电路，设计实现数据宽度为8位、最大长度为16的FIFO队列，其逻辑符号如图-9所示。入队列使能（en\_in）有效时，将输入数据（din）加入队尾；出队列使能（en\_out）有效时，将队列头数据输出（dout）。队列数据计数（count）指示队列中有效数据个数。当队列满（count = 16）时不能执行入队操作，队列空（count = 0）时不能进行出队操作。在入对使能信号的一次有效持续期间，仅允许最多入队一个数据，出队操作类似。

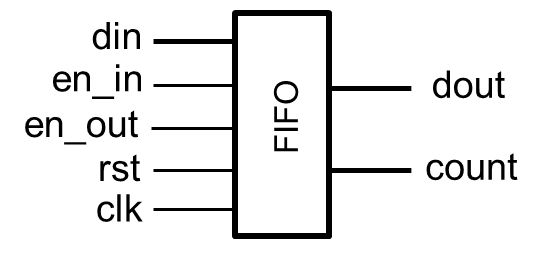


图-9 FIFO队列逻辑符号

该FIFO队列模块端口声明如下：

module fifo

(input clk, rst, //时钟（上升沿有效）、异步复位（高电平有效）

input [7:0] din, //入队列数据

input en\_in, //入队列使能，高电平有效

input en\_out, //出队列使能，高电平有效

output [7:0] dout, //出队列数据

output [4:0] count //队列数据计数

);

……

endmodule

1. 实验步骤
2. 采用行为方式描述参数化的寄存器堆，并进行功能仿真；
3. 采用IP例化方式分别实现分布式和块式的16 x 8位单端口RAM，并进行功能仿真和对比；
4. 设计FIFO队列电路的数据通路和控制器，采用结构化方式描述数据通路，两段式FSM描述控制器，并进行功能仿真；
5. 将FIFO队列电路下载至FPGA中测试：din对应SW7 ~ SW0，dout对应LED7 ~ LED0，count对应LED15~LED11，en\_in对应BTNU，en\_out对应BTND，clk对应100 MHz时钟（clk100mhz），rst对应BTNL。
6. 实验检查
7. 检查寄存器堆和RAM的功能仿真；
8. 检查FIFO队列电路的功能仿真；
9. 检查FIFO队列电路下载到FPGA后的运行功能。
10. 思考题
11. 如何利用寄存器堆和适当电路设计实现可变个数的数据排序电路？